Presentation

Agenda

Virtual Prototyping and Performance Analysis of RapidIO-based System Architectures for Space-Based Radar

David Bueno, Chris Conger, Adam Leko, Ian Troxel and ***Alan D. George***
University of Florida
Phone: 352-392-9034
Email Addresses: {bueno, conger, leko, troxel, george}@hcs.ufl.edu

Abstract

Space-Based Radar (SBR) processing is a processor- and communication-intensive HPEC application that presents unique design challenges. This talk will concentrate on the presentation of simulation results of mapping a parallel Ground Moving Target Indicator (GMTI) application on an embedded multiprocessor satellite processing system featuring a RapidIO interconnection network. We consider three partitionings of a real-time GMTI algorithm executed on systems of different sizes and topologies. Each partitioning's system performance and algorithm scalability on various RapidIO systems is examined.

Introduction

RapidIO is an emerging open standard for high-speed, embedded switched interconnection networks which supports data rates up to approximately 60 Gbps. It is an open standard [1, 2] steered by a non-profit organization known as the RapidIO Trade Association. RapidIO uses Low-Voltage Differential Signaling (LVDS) to minimize power usage at high clock speeds, and therefore is appropriate for use in HPEC systems. RapidIO is the latest commercial-off-the-shelf (COTS) technology to be considered practical for inclusion in military embedded networks to improve cost-effectiveness and scalability. Moving from bus designs to switched interconnects will substantially increase the cost-effectiveness, robustness and raw network performance of future embedded systems.

GMTI is an important application in military operations, since moving targets may be laid over a map of a battlefield for strategic planning during a conflict. GMTI works best when combined with some form of airborne radar system. Since space is the ultimate "high ground" for radar systems, having GMTI available in an SBR system is advantageous. The challenge for HPEC systems is to provide real-time data in-system with minimal latency. In traditional air-based GMTI systems, a high-performance cluster of workstations is used to process incoming radar data [3]. As GMTI requires costly adaptive processing (including Space-Time Adaptive Processing or STAP [4]) of high-resolution data, the algorithm imposes severe processing and communication challenges on space-based embedded systems with strict power, size, weight, and radiation constraints.

In order to effectively design RapidIO-based architectures, it is essential to fully understand RapidIO's strengths and weaknesses. A simulation-based testbed provides an ideal environment for performing tradeoff studies on RapidIO's salient features; therefore, we developed a simulation environment to prototype and evaluate RapidIO-based multiprocessor satellite systems for Space-Based Radar (SBR) applications within our discrete-event simulator of choice, Mission-Level Designer [5]. Our RapidIO prototyping environment incorporates moderate-fidelity systems and components including RapidIO switches, end-points and processor models. This prototyping environment has been used to predict the performance of future RapidIO space-based GMTI systems, as well as examine possible power and scalability limitations the technology may impose.

Experimental Setup

We have started with a baseline GMTI algorithm and have employed three different decomposition strategies, including a “straightforward” approach, a staggered approach, and a parallel-pipelined approach. The straightforward approach maps the incoming data set for processing equally across each of the available processors. Since the GMTI algorithm is typically composed of signal processing procedures with no interprocessor communication during each stage, the algorithm can be considered embarrassingly parallel. The staggered partitioning method is based on the approach described in [6]. This approach is similar to the straightforward mapping approach, except incoming data is sent to groups of processors in a staggered fashion. Under this approach, each processor receives a larger amount of data to process at a time, but receives this data less frequently. Our parallel-pipelined approach is a simplified version of one presented in [7], adapted to fit our vector-based processor models and RapidIO interconnection network. We split the

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **01 FEB 2005**	2. REPORT TYPE **N/A**	3. DATES COVERED -
4. TITLE AND SUBTITLE **Virtual Prototyping and Performance Analysis of RapidIO-based System Architectures for Space-Based Radar**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Florida**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR’S ACRONYM(S)
		11. SPONSOR/MONITOR’S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release, distribution unlimited**		
13. SUPPLEMENTARY NOTES **See also ADM00001742, HPEC-7 Volume 1, Proceedings of the Eighth Annual High Performance Embedded Computing (HPEC) Workshops, 28-30 September 2004 Volume 1., The original document contains color images.**		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**UU**	**25**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

pipeline into four stages, with specific groups of processors in the system dedicated to each stage.

A complete description of all system designs and network tradeoffs performed in the course of this study will appear in the full presentation. Due to page limitations, a condensed version follows. The sensors created new image data at a rate of 4.6 Gbps. Processors were modeled to perform all or a subsection of the GMTI algorithm as the partitioning warranted. RapidIO-related parameters for endpoints include a 250MHz physical-layer clock rate, input/output buffer sizes of 8 packets, and physical-layer link width of 16 bits, among others. RapidIO switch model parameters include an average memory read/write latency of 72 ns and a central memory size of 10000 bytes, among others.

Results

Figure 1 shows a summary of the results of each of the three partitioning strategies executed on systems of different sizes. The results show a system of 24 nodes is required to meet the application processing requirements of one coherent processing interval (CPI) per 256ms (denoted by the horizontal bar in the figure). The straightforward partitioning method provides the best raw performance on the RapidIO system, but the pipelining partitioning method may provide a more cost-effective strategy if individual processors for each GMTI step can be produced in a less expensive manner than an all-inclusive design. The staggered approach did not perform as well as the other two due to communication inefficiency. A broad array of additional results describing system design tradeoffs will be included in the final presentation but are omitted here due to space limitations.

Conclusions

The inclusion of RapidIO in future satellite payload processing systems is likely to improve performance as well as cost effectiveness of embedded SBR platforms. In order to prototype and predict the performance of future RapidIO space-based GMTI systems, simulation models were designed and developed using the Mission-Level Designer discrete-event simulator. Several systems, RapidIO versions, and GMTI decompositions were developed on which a tradeoff analysis was performed. The results showed a 24-processor solution met the algorithm's real-time requirements. The straightforward partitioning method provides the best raw performance on the RapidIO system, but the pipelining partitioning method may provide a more cost-effective strategy for some projects. RapidIO and other COTS-based switched interconnect designs have the potential to outperform traditional bus designs in embedded systems.

Future directions for this work may include mapping other SBR algorithms with different processing characteristics such as Synthetic Aperture Radar (SAR). In addition, now that we have an initial prototyping environment developed, we plan to examine other RapidIO-specific design considerations and system development options.

Acknowledgements

We wish to thank Honeywell Space Systems in Clearwater, FL for their funding and technical guidance in support of this research.

References

[1] "RapidIO Interconnect Specification (Parts I-IV)," RapidIO Trade Association, June 2002.

[2] "RapidIO Interconnect Specification, Part VI: Physical Layer 1x/4x LP-Serial Specification." RapidIO Trade Association, June 2002.

[3] M. Linderman and R. Linderman, "Real-Time STAP Demonstration on an Embedded High Performance Computer," Proc. of the IEEE National Radar Conference, Syracuse, NY, May 13-15, 1997.

[4] "Space-Time Adaptive Processing for Air-borne Radar," Tech. Rep. 1015, MIT Lincoln Laboratory, 1994.

[5] G. Schorcht, I. Troxel, K. Farhangian, P. Unger, D. Zinn, C. Mick, A. George, and H. Salzwedel, "System-Level Simulation Mod-eling with MLDesigner," Proc. of 11th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Com-puter and Telecommunications Systems (MASCOTS), Orlando, FL, October 12-15, 2003.

[6] R. Brown and R. Linderman, "Algorithm De-velopment for an Airborne Real-Time STAP Demonsttration," Proc. of the IEEE National Radar Conference, Syracuse, NY, May 13-15, 1997.

[7] A. Choudhary, W. Liao, D. Weiner, P. Varsh-ney, R. Linderman, M. Linderman, and R. Brown, "Design, Implementation and Evalua-tion of Parallel Pipelined STAP on Parallel Computers," *IEEE Trans. on Aerospace and Electrical Systems*, vol. 36, pp 528-548, April 2000.

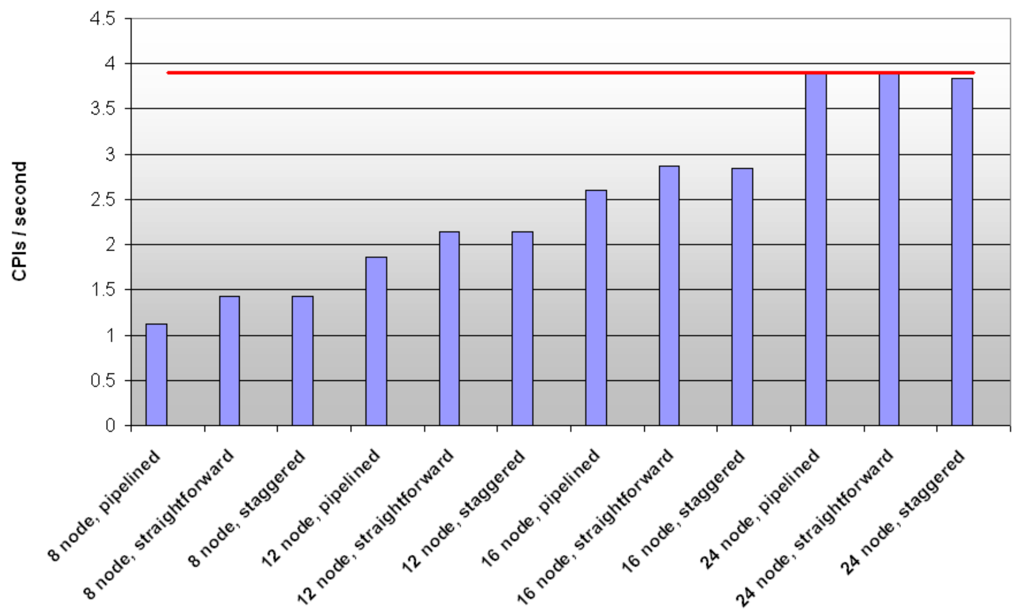


Figure 1: System throughput

Virtual Prototyping and Performance Analysis of RapidIO-based System Architectures for Space-Based Radar

David Bueno, Adam Leko, Chris Conger,
Ian Troxel, and Alan D. George

HCS Research Laboratory
College of Engineering
University of Florida

28 September 2004

Outline

Project Overview

- Simulative analysis of Space-Based Radar (SBR) systems using RapidIO interconnection networks
  - RapidIO (RIO) is a high-performance, switched interconnect for embedded systems
    - Can scale to many nodes
    - Provides better bisection bandwidth than existing bus-based technologies

- Study optimal method of constructing scalable RIO-based systems for Ground Moving Target Indicator (GMTI)
  - Identify system-level tradeoffs in system designs
  - Discrete-event simulation of RapidIO network, processing elements, and GMTI algorithm
  - Identify limitations of RIO design for SBR
  - Determine effectiveness of various GMTI algorithm partitionings over RIO network

Image courtesy [1]

Background- RapidIO

- Three-layered, embedded system interconnect architecture
  - Logical – memory mapped I/O, message passing, and globally shared memory
  - Transport
  - Physical – serial and parallel
- Point-to-point, packet-switched interconnect
- Peak single-link throughput ranging from 2 to 64 Gb/s
- Focus on 16-bit parallel LVDS RIO implementation for satellite systems

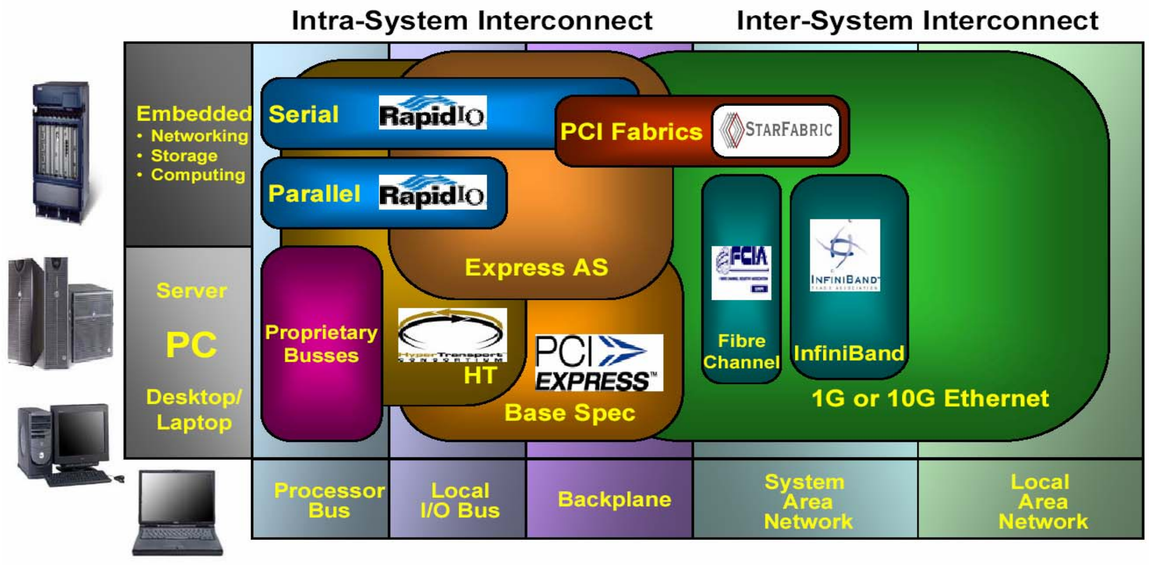
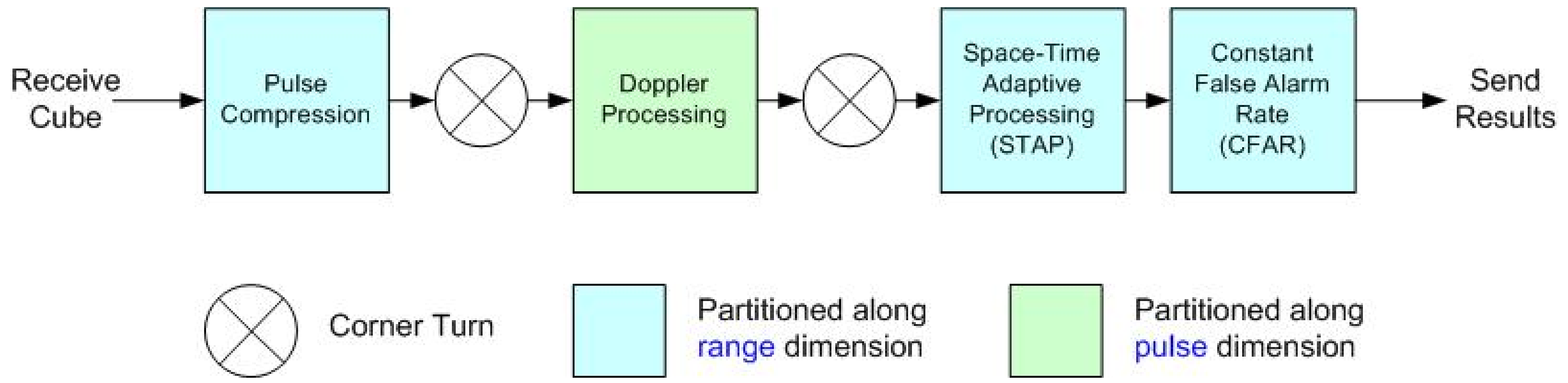


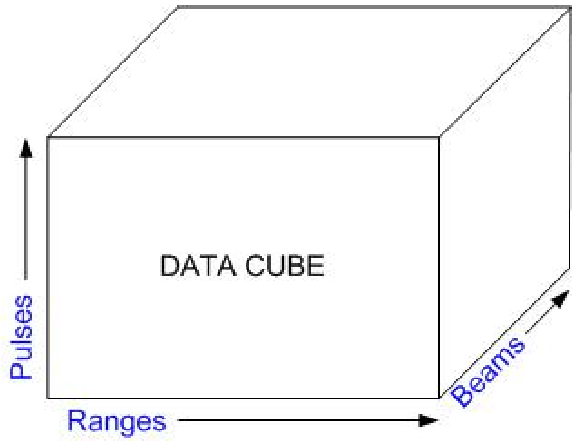
Image courtesy [2]

Background- GMTI



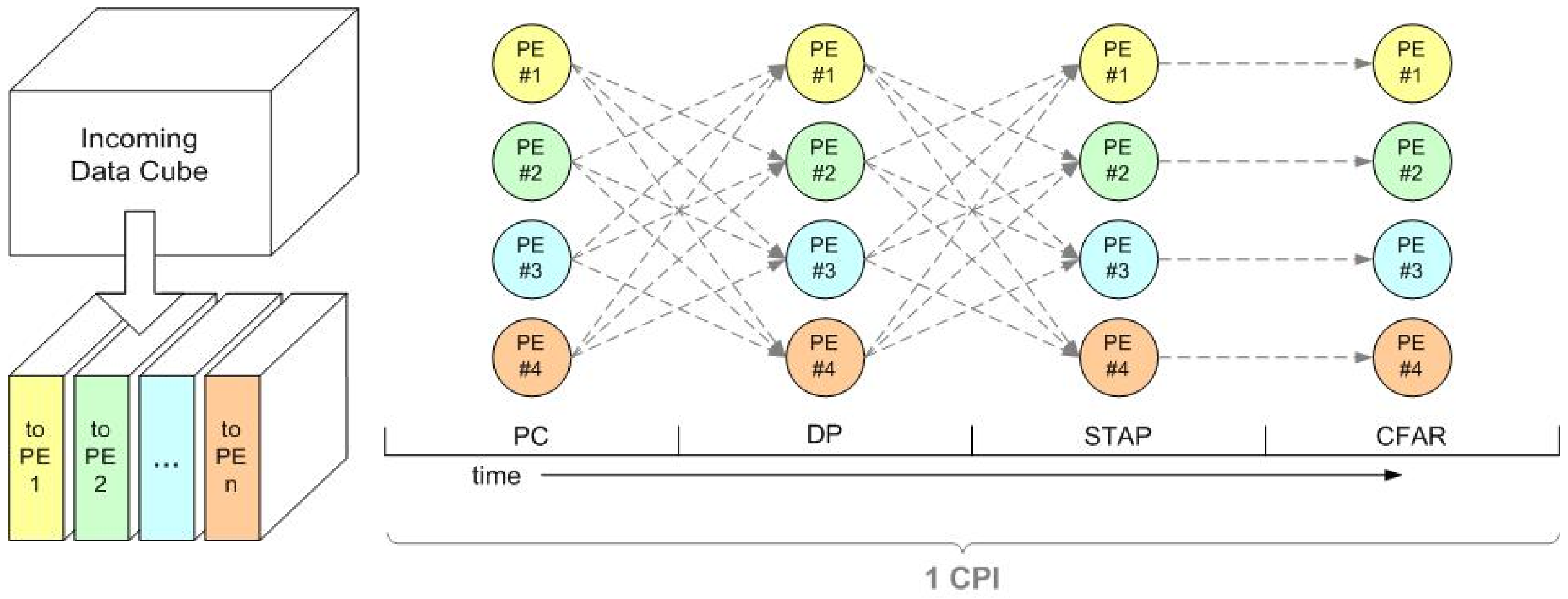
Receive Cube
Pulse Compression
Doppler Processing
Space-Time Adaptive Processing (STAP)
Constant False Alarm Rate (CFAR)
Send Results
Corner Turn
Partitioned along range dimension
Partitioned along pulse dimension

- GMTI used to track moving targets on ground
  - Estimated processing requirements range from 40 (aircraft) to 280 (satellite) GFLOPs
- GMTI broken into four stages:
  - Pulse Compression (PC)
  - Doppler Processing (DP)
  - Space-Time Adaptive Processing (STAP)
  - Constant False-Alarm Rate detection (CFAR)



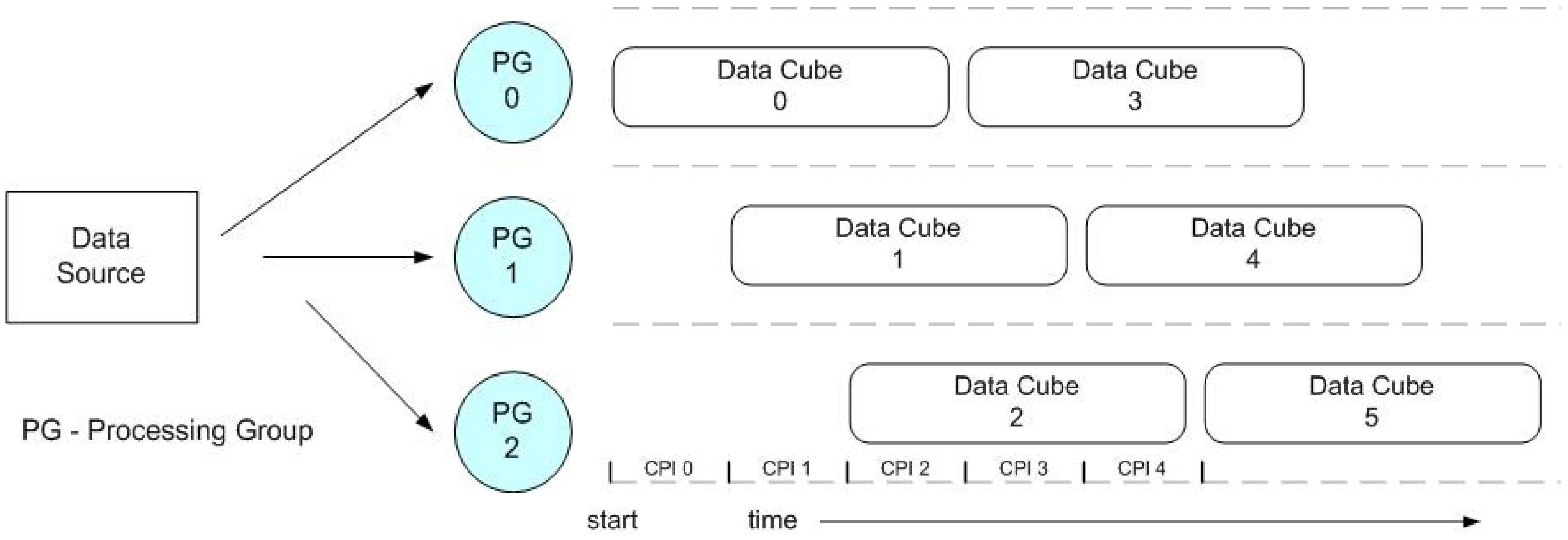
- Incoming data organized as 3-D matrix (data cube)
  - Data reorganization ("corner turn") necessary between stages for processing efficiency
  - Size of each cube dictated by Coherent Processing Interval (CPI)

GMTI Partitioning Methods- Straightforward



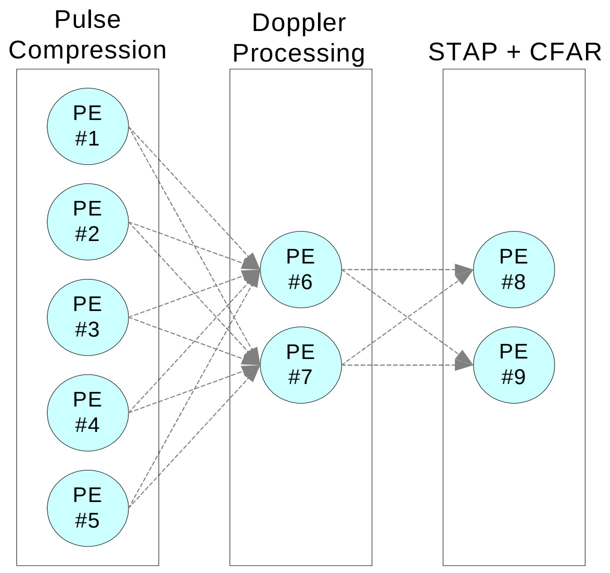
- Data cubes divided among all Processing Elements (PEs)
- Partitioned along optimal dimension for any particular stage
- Data reorganization between stages implies personalized all-to-all communication (corner turn) $\Rightarrow$ stresses backplane links
- Minimal latency
  - Entire cube must be processed within one CPI to receive next cube

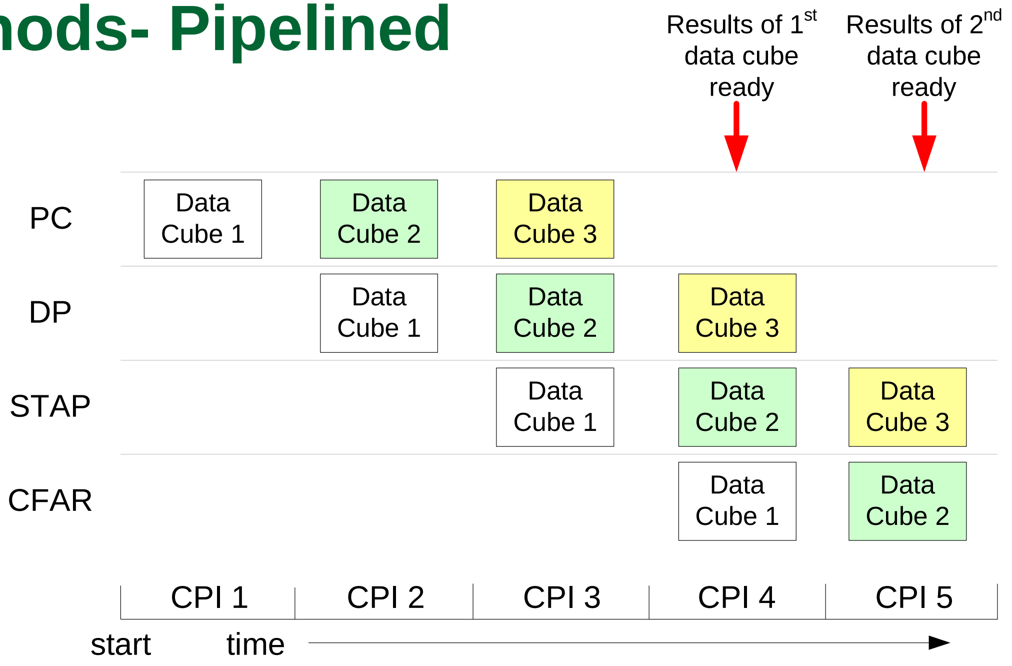
GMTI Partitioning Methods- Staggered



- **Data cubes sent to groups of PEs in round-robin fashion**
  - **Limiting each Processing Group (PG) to a single board significantly reduces backplane bandwidth impact**
- **Time given to each PG to receive and process a data cube is N × CPI**
  - **N = number of processing groups**
  - **CPI = amount of time between generated data cubes**
- **Latency to produce result is higher than in straightforward partitioning**

GMTI Partitioning Methods- Pipelined



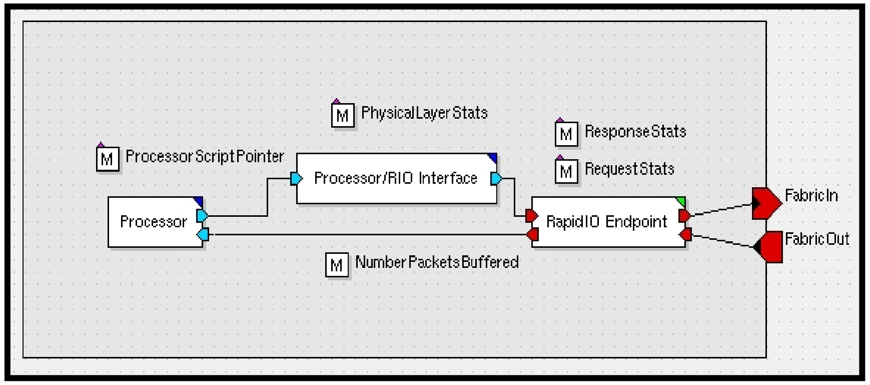


- **Each PE group assigned to process a single stage of GMTI**
  - **Groups may have varying numbers of PEs depending upon processing requirements of each stage**
- **Potential for high cross-system bandwidth requirements**
  - **Irregular and less predictable traffic distribution**
  - **Frequent communication between different group sizes**
- **Latency to produce result is higher than straightforward method**
  - **One result emerges each CPI, but the results are three CPIs old**

Model Library Overview

- Modeling library created using Mission Level Designer (MLD), a commercial discrete-event simulation modeling tool
  - C++-based, block-level, hierarchical modeling tool
- Algorithm modeling accomplished via script-based processing
  - All processing nodes read from a global script file to determine when/where to send data, and when/how long to compute
- Our model library includes:
  - RIO central-memory switch
  - Compute node with RIO endpoint
  - GMTI traffic source/sink
  - RIO logical message-passing layer
  - Transport and parallel physical layers

Model of Compute Node with RIO Endpoint

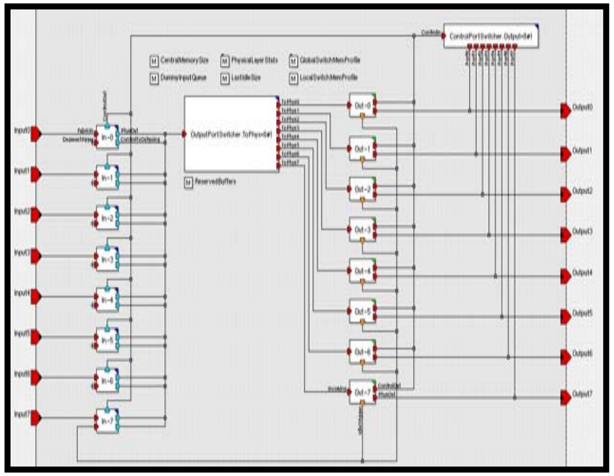


RapidIO Models

RapidIO The Embedded System Interconnect

- Key features of Endpoint model
  - Message-passing logical layer
  - Transport layer
  - Parallel physical layer
    - Transmitter- and receiver-controlled flow control
    - Error detection and recovery
    - Priority scheme for buffer management
    - Adjustable link speed and width
    - Adjustable priority thresholds and queue lengths
- Key features of Central-memory switch model
  - Selectable cut-through or store-and-forward routing
  - High-fidelity TDM model for memory access
  - Adjustable priority thresholds based on free switch memory
  - Adjustable link rates, etc. similar to endpoint model

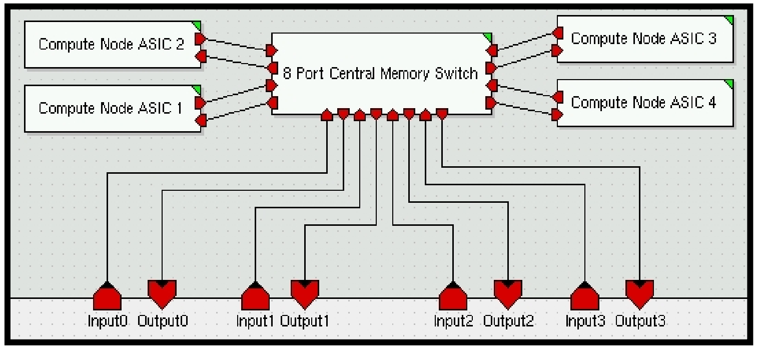
Model of RIO Central-Memory Switch

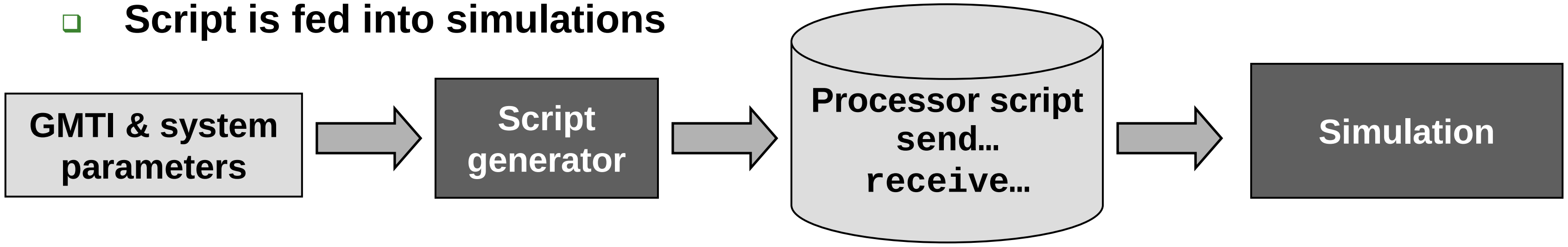


GMTI Processor Board Models

- System contains many processor boards connected via backplane
- Each processor board contains one RIO switch and four processors
- Processors modeled with three-stage finite state machine
  - Send data
  - Receive data
  - Compute
- Behavior of processors controlled with script files
  - Script generator converts high-level GMTI parameters to script
  - Script is fed into simulations

Model of Four-Processor Board



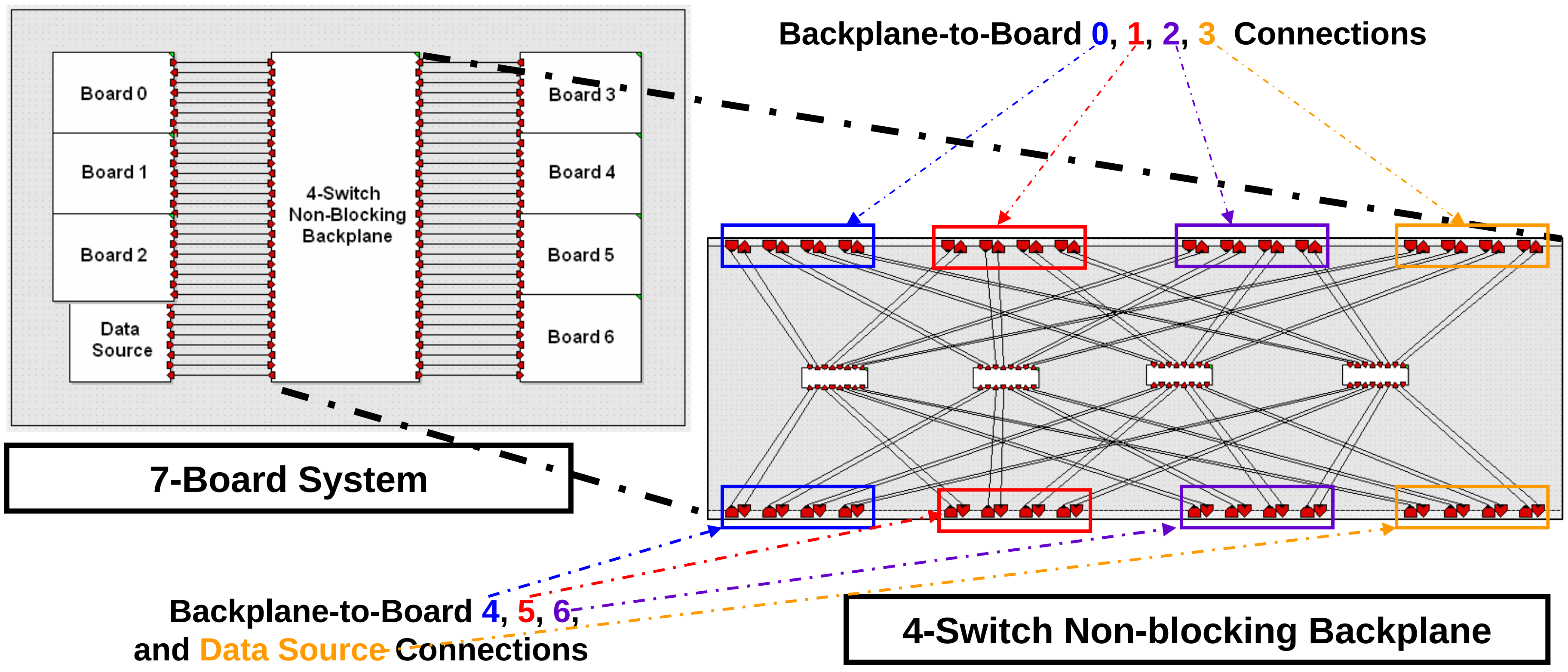


System Design Constraints

- 16-bit parallel 250MHz DDR RapidIO links (1 GB/s)
  - Expected radiation-hardened component performance by time RIO and SBR ready to fly in ~2008 to 2010
- Systems composed of processor boards interconnected by RIO backplane
  - 4 processors per board
  - 8 Floating-Point Units (FPUs) per processor
  - One 8-port central-memory switch per board; implies 4 connections to backplane per board
- Baseline GMTI algorithm parameters:
  - Data cube: 64k ranges, 256 pulses, 6 beams
  - CPI = 256ms
  - Requires ~3 GB/s of aggregate throughput from source to sink to meet real-time constraints

Backplane and System Models

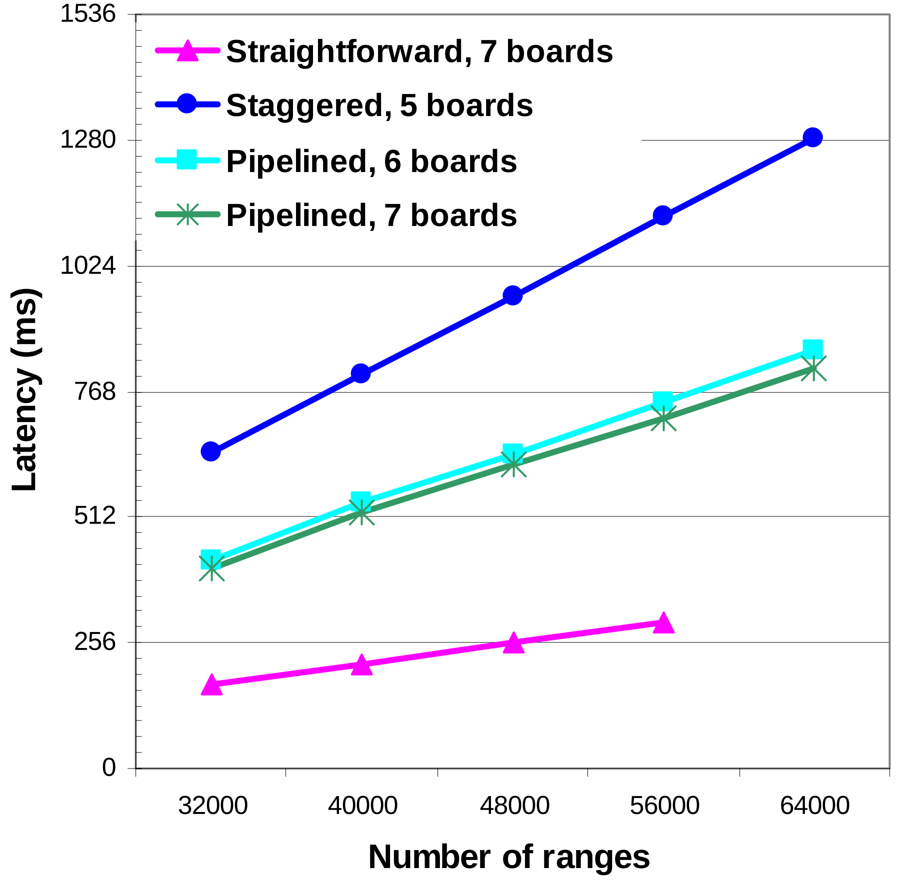
- **High throughput requirements for data source and corner turns require non-blocking connectivity between all nodes and data sources**


Board 0
Board 1
Board 2
Data Source
4-Switch Non-Blocking Backplane
Board 3
Board 4
Board 5
Board 6
7-Board System
Backplane-to-Board 0, 1, 2, 3 Connections
Backplane-to-Board 4, 5, 6, and Data Source Connections
4-Switch Non-blocking Backplane

Overview of Experiments

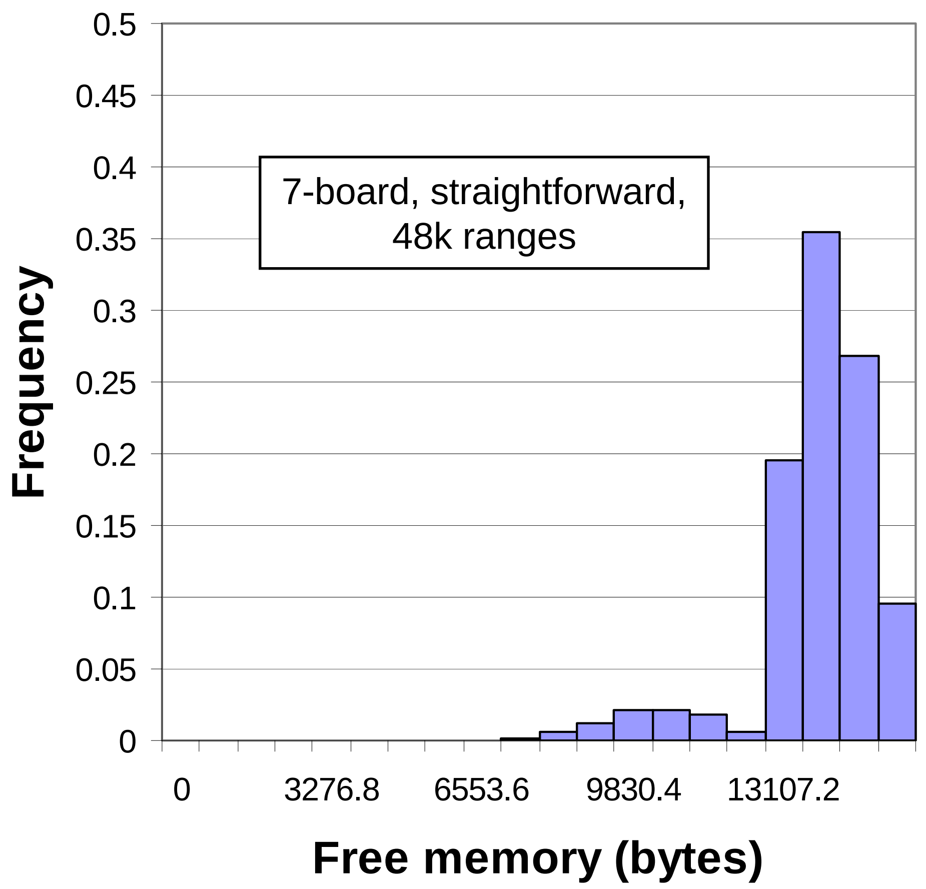
- Experiments conducted to evaluate strengths and weaknesses of each partitioning method
- Same switch backplane used for each experiment
- Varied data cube size
  - 256 pulses, 6 beams for all tests
  - Varied number of ranges from 32k to 64k
- Several system sizes used
  - Analysis determined that 7-board configuration necessary for straightforward method to meet deadline
  - Both 6- and 7-board configurations used for pipelined method
  - Staggered method does not benefit from a system larger than 5 boards with configuration used
    - Staggering performed with one processor board per group
    - Larger system-configurations leave processors idle

Result Latency Comparison



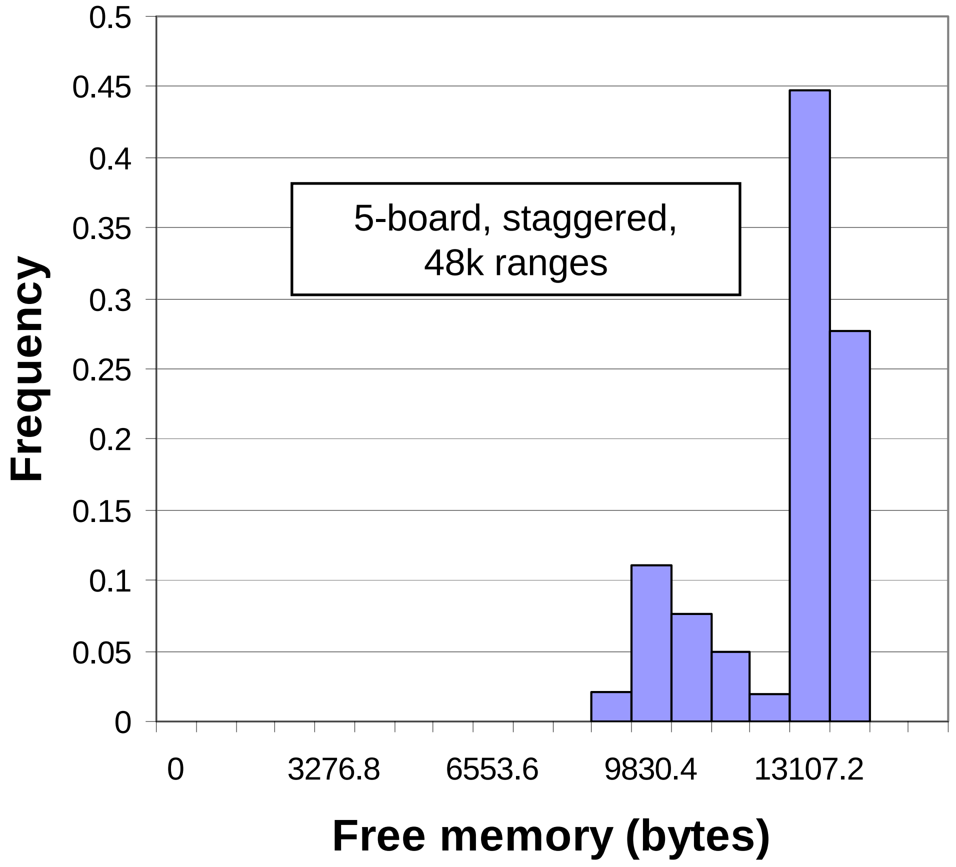
- Result latency is interval from data arrival until results reported
- Straightforward achieved lowest latency, required most processor boards
  - No result for 64k ranges because system could not meet real-time deadline
- Staggered requires least number of processor boards to meet deadline
  - Efficient system configuration, small communication groups
  - Tradeoff is result latency
- Pipelined method a compromise

Switch Memory Histogram with Straightforward Method



- Chart shows frequency of time free switch memory lies in each bracket
- Max switch memory is 16384 bytes
- Results taken from switch on processor board 1
  - All processor board switches see essentially identical memory usage
- ~90% of time is spent with switch ~80% free
  - Most predictable communication patterns, enabling effective static planning of comm. paths

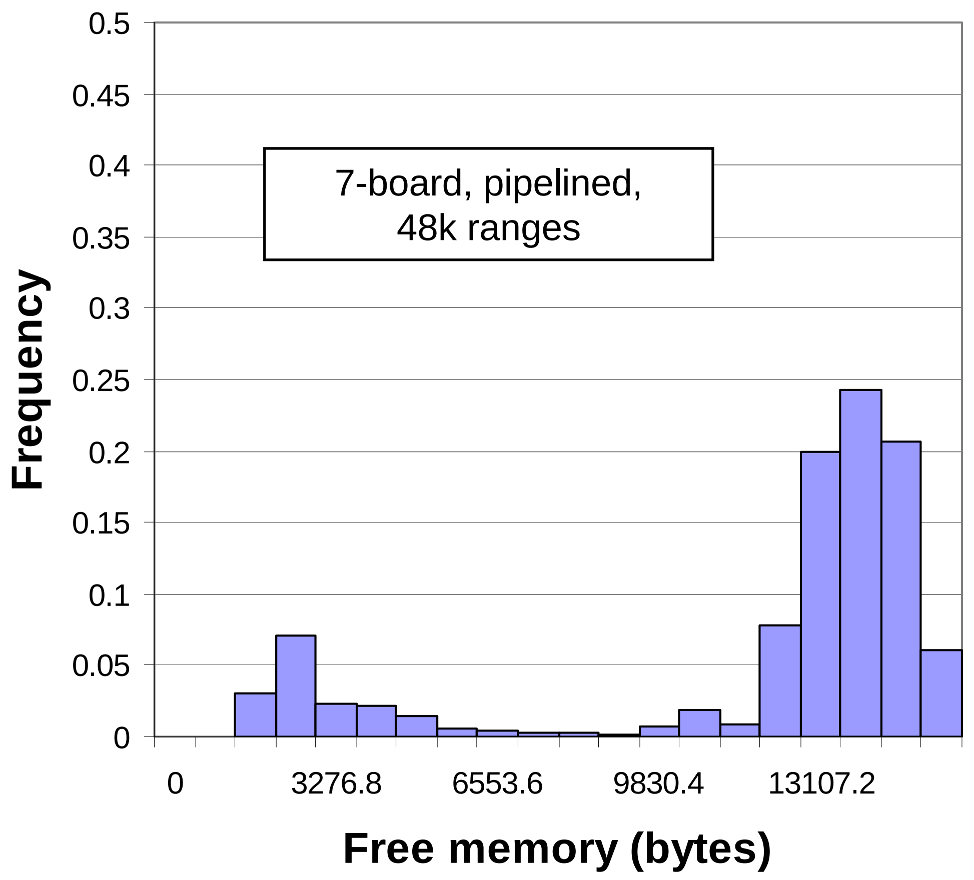
Switch Memory Histogram with Staggered Method



- Staggered method uses slightly more memory over course of simulation
  - More data flows through single switch during corner turn
  - Less spread in communication patterns than straightforward method
- More switch memory usage indicates more contention for a particular port, not necessarily more utilization or communication

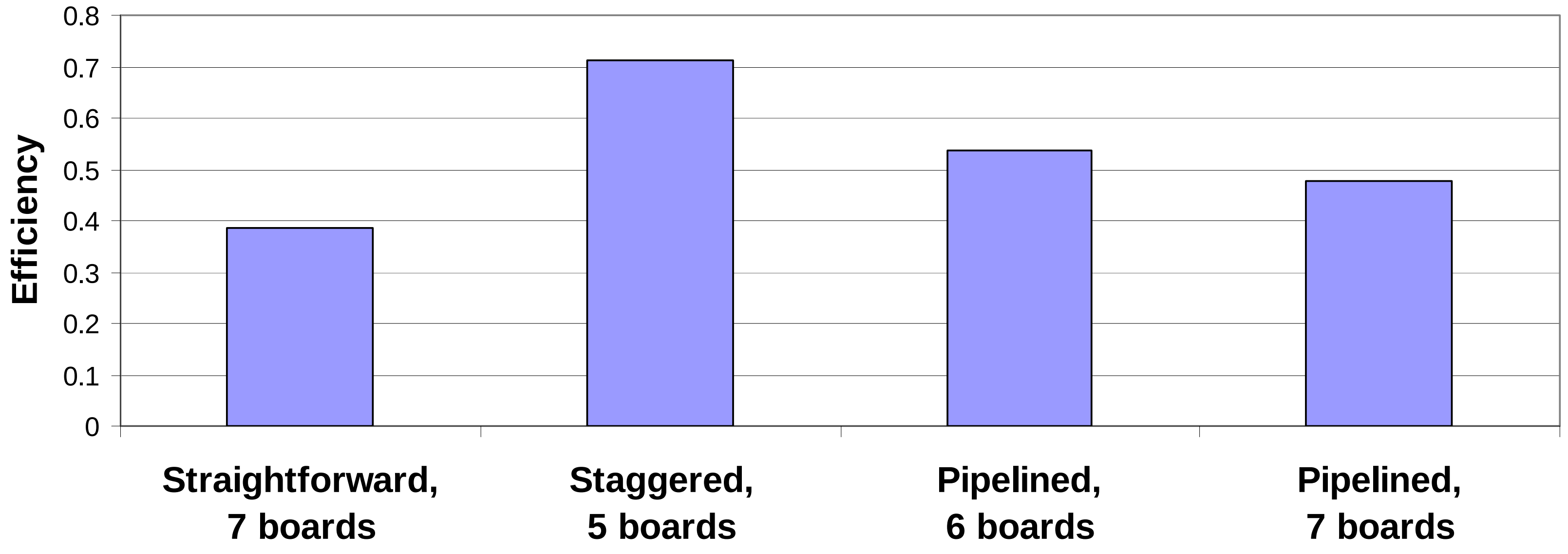
www.hcs.ufl.edu
High-performance Computing & Simulation Research Lab

Switch Memory Histogram with Pipelined Method



- **Pipelined method stresses network**
  - **Irregular comm. patterns**
  - **Greater possibility for output port contention**
  - **Non-blocking network not helpful when multiple senders vying for same destination**
- **Difficult to plan out optimal comm. paths beforehand**
  - **Much synchronization required to stagger many-to-one communication, but not extremely costly in total execution time**

Average Parallel Efficiency



- **Parallel efficiency defined as sequential execution time (i.e. result latency) divided by *N* times the parallel execution time**
  - ***N* = number of processors that work on a single CPI**
  - **Pipelined efficiency a special case, must use *N*/3 for fair comparison (shown) since all processors do not work on a CPI at the same time**
- **Staggered method most efficient due to small communication groups and low number of processors working on same CPI**
  - **Straightforward method worst for opposite reason, pipelined method a compromise**

Conclusions

- Developed suite of simulation models and mechanisms for evaluation of RapidIO designs for space-based radar
- Evaluated three partitioning methods for GMTI over a fixed RapidIO non-blocking network topology
- Straightforward partitioning method produced lowest result latencies, but least scalable
  - Unable to meet real-time deadline with our maximum data cube size
- Staggered partitioning method produced worst result latencies, but highest parallel efficiency
  - Also able to perform algorithm with least number of processing boards
  - Important for systems where power consumption, weight are a concern
- Pipelined partitioning method is a compromise in terms of latency, efficiency, and scalability, but heavily taxes network
- RapidIO provides feasible path to flight for space-based radar
  - Future work to focus on additional SBR variants (e.g. Synthetic Aperture Radar) and experimental RIO analysis

Bibliography

[1] http://www.afa.org/magazine/aug2002/0802radar.asp

[2] G. Shippen, “RapidIO Technical Deep Dive 1: Architecture & Protocol,” Motorola Smart Network Developers Forum, 2003.

[3] “RapidIO Interconnect Specification (Parts I-IV), ” RapidIO Trade Association, June 2002.

[4] “RapidIO Interconnect Specification, Part VI: Physical Layer 1x/4x LP-Serial Specification,” RapidIO Trade Association, June 2002.

[5] M. Linderman and R. Linderman, “Real-Time STAP Demonstration on an Embedded High Performance Computer,” Proc. of the IEEE National Radar Conference, Syracuse, NY, May 13-15, 1997.

[6] “Space-Time Adaptive Processing for Airborne Radar,” Tech. Rep. 1015, MIT Lincoln Laboratory, 1994.

[7] G. Schorcht, I. Troxel, K. Farhangian, P. Unger, D. Zinn, C. Mick, A. George, and H. Salzwedel, “System-Level Simulation Modeling with MLDesigner,” Proc. of 11th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS), Orlando, FL, October 12-15, 2003.

[7] R. Brown and R. Linderman, “Algorithm Development for an Airborne Real-Time STAP Demonsttration,” Proc. of the IEEE National Radar Conference, Syracuse, NY, May 13-15, 1997.

[8] A. Choudhary, W. Liao, D. Weiner, P. Varshney, R. Linderman, M. Linderman, and R. Brown, “Design, Implementation and Evaluation of Parallel Pipelined STAP on Parallel Computers,” *IEEE Trans. on Aerospace and Electrical Systems*, vol. 36, pp 528-548, April 2000.

Acknowledgements

- We wish to thank Honeywell Space Systems in Clearwater, FL for their funding and technical guidance in support of this research.
- We wish to thank MLDesign Technologies in Palo Alto, CA for providing us the MLD simulation tool that made this work possible.